APPENDIX

09451090.113099

RBAC on the Web by Secure Cookies

Joon S. Park, Ravi Sandhu, and SreeLatha Ghanta
The Laboratory for Information Security Technology
Information and Software Engineering Department
George Mason University
{jpark, sandhu, sghanta}@list.gmu.edu

ABSTRACT: Current approaches to access control on Web servers do not scale to enterprise-wide systems, since they are mostly based on individual users. Therefore, we were motivated by the need to manage and enforce the strong access control technology of RBAC in large-scale Web environments. RBAC is a successful technology that will be a central component of emerging enterprise security infrastructures. *Cookies* can be used to support RBAC on the Web, holding users' role information. However, it is insecure to store and transmit sensitive information in cookies. Cookies are stored and transmitted in clear text, which is readable and easily forged.

In this paper, we describe an implementation of Role-Based Access Control with role hierarchies on the Web by secure cookies. Since a user's role information is contained in a set of secure cookies and transmitted to the corresponding Web servers, these servers can trust the role information in the cookies after cookie-verification procedures and use it for role-based access control. In our implementation, we used CGI scripts and PGP (Pretty Good Privacy) to provide security services to secure cookies. The approach is transparent to users and applicable to existing Web servers and browsers.

KEYWORDS: Cookie, Role-Based Access Control (RBAC), WWW Security

09451.061399

1 Introduction

WWW is commonplace. Increased integration of Web, operating system, and database system technologies will lead to continued reliance on Web technology for enterprise computing. However, current approaches to access control on Web servers are mostly based on individual users; therefore, they do not scale to enterprise-wide systems.

A successful marriage of the Web and a strong and efficient access control technology has potential for considerable impact on and deployment of effective enterprise-wide security in large-scale systems. Role-based access control (RBAC) [San98] is a promising technology for managing and enforcing security in large-scale enterprise-wide systems, and it will be a central component of emerging enterprise security infrastructures. We were motivated by the need to manage and enforce the strong access control technology of RBAC in large-scale Web environments.

To support RBAC on the Web, we chose a relatively mature technology, cookies - widely used on the Web - and have extended it for our purpose. Cookies were invented to maintain continuity and state on the Web [KM97, KM98]. Cookies contain strings of text characters encoding relevant information about the user. Cookies are sent to the user's memory via the browser while the user is visiting a cookie-using Web site, and are stored on the user's hard disk after the browser is closed. The Web server gets those cookies back and retrieves the user's information from the cookies when the user later returns to the same Web site or domain. The purpose of a cookie is to acquire information and use it in subsequent communications between the Web server and the browser without asking for the same information again. Often a Web server sets a cookie to hold a pointer, such as an identification number, as a user-specific primary key of the information database maintained in the server. Technically, it is not difficult to make a cookie carry relevant information. For instance, a merchant Web server could use a cookie containing the user's name and credit card number. This is convenient for users, since they do not have to read lengthy numbers from their cards and key these in for every transaction. However, it is not safe to store and transmit this sensitive information in cookies because cookies are insecure. Cookies are stored and transmitted in clear text, which is readable and easily forged. Therefore, we should render secure cookies to carry and store sensitive data in them.

We will provide secure cookies with three types of security services: authentication, integrity, and confidentiality. Authentication services verify the owner of the cookies. Integrity services protect cookies against the threat that the contents of the cookies might be changed by unauthorized modification. Finally, confidentiality services protect cookies against the values of the cookies being revealed to an unauthorized entity. Details for these techniques have varying degrees of security and convenience for users and system administrators[1].

In this paper, we will describe how we implemented RBAC with role hierarchy [FCK95, SCFY96] on the Web using the secure cookies. To provide security services to secure cookies, we used CGI scripts and the PGP (Pretty Good Privacy) package, which are already in widespread current use.

The rest of this paper is organized as follows. First, in Section 2, we describe the technologies most relevant to our work, such as RBAC, cookies, and PGP. In Section 3, we describe security threats to cookies, and how to design secure cookies to support RBAC on the Web. In Section 4, we describe how we actually implemented RBAC on the Web by secure cookies, using CGI scripts and PGP on an existing Web server. Section 5 gives our conclusions.

2 Related Technologies

2.1 Role-Based Access Control (RBAC)

Role-based access control (RBAC) [San98] has rapidly emerged in the 1990s as a promising technology for managing and enforcing security in large-scale enterprise-wide systems. The

[1]For secure communications on the Web, we may consider using other existing technologies, such as, SHTTP (Secure HTTP [RS98, SR98]) and SSL (Secure Socket Layer [WS96]). However, these technologies cannot solve the *stateless* problem of HTTP. Furthermore, none of these can prevent end-system threats to cookies.

09451069 . 110399

basic notion of RBAC is that permissions are associated with roles, and users are assigned to appropriate roles. This greatly simplifies security management.

A role is a semantic construct forming the basis of access control policy. With RBAC, system administrators can create roles, grant permissions to those roles, and then assign users to the roles on the basis of their specific job responsibilities and policy. Therefore, role-permission relationships can be predefined, which makes it simple to assign users to the predefined roles. Without RBAC, it is difficult to determine what permissions have been authorized for what users.

RBAC is a promising alternative to traditional discretionary and mandatory access controls, and ensures that only authorized users are given access to certain data or resources. It also supports three well-known security policies: data abstraction, least-privilege assignment, and separation of duties.

2.2 Cookies

At present, there are many browsers that support cookies, including Netscape, MS Internet Explorer, GNNWorks, NetCruiser and OmniWeb. Cookies have been used for many purposes on the Web, such as selecting display mode (e.g., frames or text only), shopping cart selections, and carrying names, passwords, account numbers, or some other bits of identifying data on the user's machine. Although there are many ways to use cookies on the Web, the basic process and the contents of cookies are similar. The detailed cookie specifications are available in [KM97, KM98].

Whenever a browser sends an HTTP request for a URL to a Web server, only those cookies relevant to that server will be sent by the browser. If the server finds any cookies that are related to the server, those cookies are used during this communication between the browser and the server. However, if the server does not find any cookies specified for it, either that server does not use cookies in the communication or the server creates new cookies for subsequent communication between the browser and the server.

Web servers may update the contents of their cookies for any specific circumstance. The cookie-issuer is not important for cookie validation. In other words, a server can create cookies for other servers in the domain. This is an important aspect of cookies that will be used in our implementations described in Section 4.

2.3 Pretty Good Privacy (PGP)

PGP (Pretty Good Privacy), a popular software package originally developed by Phil Zimmermann, is widely used by the Internet community to provide cryptographic routines for e-mail, file transfer, and file storage applications [Zim95]. A proposed Internet standard has been developed [CDFT98], specifying use of PGP. It uses existing cryptographic algorithms and protocols and runs on multiple platforms. It provides data encryption and digital signature functions for basic message protection services.

PGP is based on public-key cryptography. It defines its own public-key pair management system and public-key certificates. The PGP key management system is based on the relationship between key owners, rather than on a single infrastructure such as X.509.

09511095.113099

Basically, it uses RSA for the convenience of the public-key cryptosystem, message digests (MD5, IDEA) for the speed of process, and Diffie-Hellman for key exchange. The updated version supports more cryptographic algorithms.

Even though the original purpose of PGP is to protect casual e-mail between Internet users, we decided to use the PGP package. The package is already widely used and satisfies our requirements, in conjunction with Web servers via CGI scripts for our implementation to protect cookies. These cookies have role information of the user.

3 Secure Cookies

3.1 Security Threats to Cookies

We distinguish three types of threats to cookies: *network security threats*, *end-system threats* and *cookie-harvesting threats*. Cookies transmitted in clear text on the network are susceptible to snooping (for subsequent replay) and to modification by network threats. Network threats can be foiled by use of the Secure Sockets Layer (SSL) protocol [WS96] which is widely deployed in servers and browsers.[2] However, SSL can only secure cookies while they are on the network. Once the cookie is in the browser's end system it resides on the hard disk or memory in clear text. Such cookies can be trivially altered and can be easily copied from one computer to another, with or without connivance of the user on whose machine the cookie was originally stored. We call this the end-system threat. The ability to alter cookies allows users to forge authorization information in cookies and to impersonate other users. The ability to copy cookies makes such forgery and impersonation all the easier. Additionally, if an attacker collects cookies by impersonating a site that accepts cookies from the users (who believe that they are communicating with a legitimate Web server), later he can use those harvested cookies for all other sites that accept those cookies. We call this the cookie-harvesting threat. These attacks are all relatively easy to carry out and certainly do not require great hacker expertise.

3.2 Designing Secure Cookies for RBAC on the Web

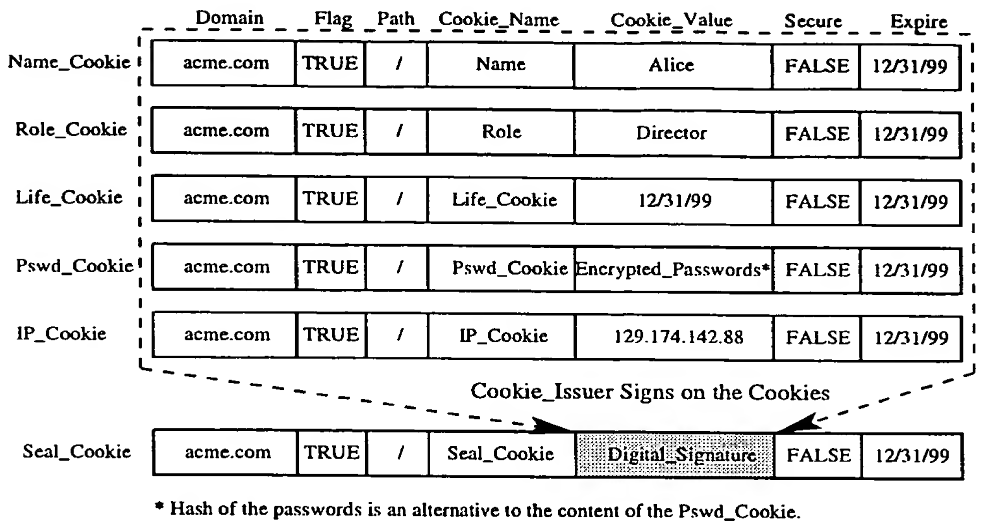
In this subsection, we describe how to transform regular cookies - which have zero security - into secure cookies, which provide the classic security services against the three types of threats to cookies (described in the previous subsection).

Secure cookies provide three types of security services: *authentication*, *integrity*, and *confidentiality services*. Selection of the kinds and contents of secure cookies depends on applications and a given situation. However, at least one authentication cookie and the Seal_Cookie - which provides the integrity service to the cookies - must be used with other cookies to frame basic security services, regardless of applications.

Figure 1 shows a set of secure cookies that we will create and use for RBAC on the Web. The Name_Cookie contains the user's name (e.g., Alice), and the Role_Cookie holds

[2]In many cases due to export restrictions from USA only weak keys (40 bits) are supported, but SSL technology is intrinsically capable of very strong protection against network threats.

09354104.111301



	Domain	Flag	Path	Cookie_Name	Cookie_Value	Secure	Expire
Name_Cookie	acme.com	TRUE	/	Name	Alice	FALSE	12/31/99
Role_Cookie	acme.com	TRUE	/	Role	Director	FALSE	12/31/99
Life_Cookie	acme.com	TRUE	/	Life_Cookie	12/31/99	FALSE	12/31/99
Pswd_Cookie	acme.com	TRUE	/	Pswd_Cookie	Encrypted_Passwords*	FALSE	12/31/99
IP_Cookie	acme.com	TRUE	/	IP_Cookie	129.174.142.88	FALSE	12/31/99
Seal_Cookie	acme.com	TRUE	/	Seal_Cookie	Digital_Signature	FALSE	12/31/99

* Hash of the passwords is an alternative to the content of the Pswd_Cookie.

Figure 1: A Set of Secure Cookies for RBAC on the Web

09451090 . 11309

the user's role information (e.g., Director). The Life_Cookie is used to hold the lifetime of the secure-cookie set in its Cookie_Value field and enables the Web server to check the integrity of the lifetime of the secure-cookie set. To protect these cookies from possible attacks, we will use IP_Cookie, Pswd_Cookie, and Seal_Cookie. Authentication cookies (i.e., IP_Cookie and Pswd_Cookie) verify the owner of the cookies by comparing the authentication information in the cookies to those coming from the users. The IP_Cookie holds the IP number of the user's machine, and the Pswd_Cookie holds the user's encrypted passwords. This confidentiality service protects the values of the cookies from being revealed to unauthorized entity. In our implementation, we used the IP_Cookie and Pswd_Cookie together to show the feasibility, but only one of those authentication cookies can be used to provide the authentication service. The choice of an authentication cookie depends on the situation.[3] Finally, the Seal_Cookie - which has the digital signature of the cookie-issuing server on the secure cookie set - supports integrity service, protecting cookies against the threat that the contents of the cookies might be changed by unauthorized modification.

There are basically two cryptographic technologies applicable for secure cookies: public-key-based and secret-key-based solutions. In our implementation, we use the public-key-based solution for security services provided by a PGP package via CGI scripts. In the next section, we will describe secure cookie creation, verification, and use of the role information in the Role_Cookie for RBAC with role hierarchies, in turn.

[3]It is also possible for authentication to be based on use of RADIUS [RRSW97], Kerberos [SNS88, Neu94], and similar protocols. Our focus in this work is on techniques that make secure cookies self-sufficient rather than partly relying on other security protocols, which is always possible.

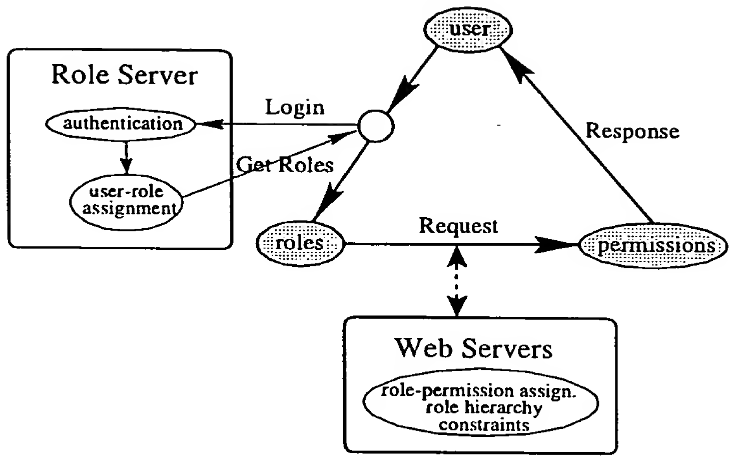


Figure 2: A Schematic of RBAC on the Web

09451090 .113009

4 RBAC Implementation by Secure Cookies

Figure 2 shows a schematic of RBAC on the Web. The role server has user-role assignment information for the domain. After a successful user authentication, the user receives his or her assigned roles in the domain from the role server. Later, when the user requests access to a Web server with the assigned roles in the domain, the Web server allows the user to execute transactions based on the user's roles instead of identity. The Web servers may have role hierarchies or constraints based on their policies.

However, how can the Web servers trust the role information presented by users? For instance, a malicious user may have unauthorized access to the Web servers by using forged role information. Therefore, we must protect the role information from being forged by any possible attacks on the Web as well as in the end-systems.

There can be many possible ways to support the above requirement. In this paper, as one possible solution, we will describe how to protect the role information from possible threats using secure cookies, and how we implemented RBAC (Role-Based Access Control) with role hierarchy on the Web. Figure 3 shows how the secure cookies (including a Role_Cookie) for RBAC are created and used on the Web. If a user, let's say Alice, wants to execute transactions in the Web servers in an RBAC-compliant domain, she first connects to the role server in the beginning of the session. After the role server authenticates Alice, it finds Alice's explicitly assigned roles in the URA (User-Role Assignment [SP98, SB97]) database and creates a set of secure cookies: Name_Cookie, Life_Cookie, Role_Cookies, IP_Cookie, Pswd_Cookie, and Seal_Cookie. Then, those secure cookies are sent to and stored in Alice's hard drive securely so that Alice does not need to go back to the role server to get her assigned roles until the cookies expire. Namely, she can use the roles in her Role_Cookie securely in the RBAC-compliant domain as long as the cookies are valid.

When Alice requests access to a Web server - which has PRA (Permission-Role As-

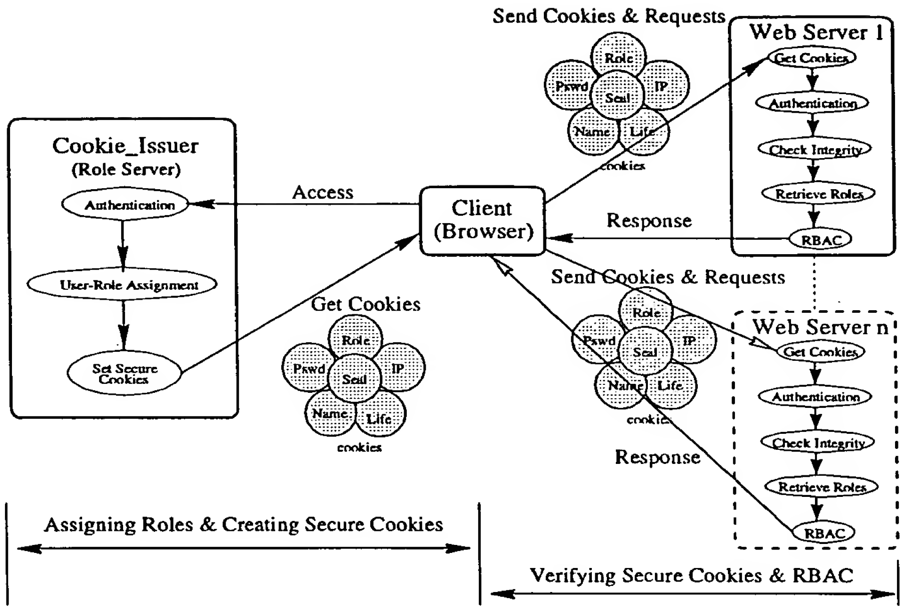


Figure 3: RBAC on the Web by Secure Cookies

0945109.11309

signment [SBC+97]) information - by typing the server URL in her browser, the browser sends the corresponding set of secure cookies to the Web server: Name_Cookie, Life_Cookie, Role_Cookies, IP_Cookie, Pswd_Cookie, and Seal_Cookie. The Web server authenticates the owner of the cookies by using the IP_Cookie and Pswd_Cookie, comparing the value in the cookies with the values coming from the user. The user's passwords are encrypted in the Pswd_Cookie using the Web server's public key. The Web server decrypts the value of the Pswd_Cookie by using the corresponding key to read the user's passwords. Finally, the Web server checks the integrity of the cookies by verifying role server's digital signature in the Seal_Cookie using the role server's public key. If all the cookies are valid and verified successfully, the Web server trusts the role information in the Role_Cookie and uses it for RBAC with a role hierarchy and permission-role assignment information in the Web server.

4.1 Secure Cookie Creation

When a user, Alice, connects to the role server (which supports HTTP) of the domain with her Web browser, she is prompted by the HTML form to type in her user ID and passwords for the domain. We used the POST[4] method to send the information to the role server and the ACTION field to specify our Cookie-Set CGI program (set-cookie.cgi), to which the form data is passed. Figure 4 is a collaborational diagram in UML (Unified Modeling

[4]The GET request is very similar to the POST except that the values of the form variable are sent as part of the URL. However, the POST method sends the data after all their request headers have been sent to the server.

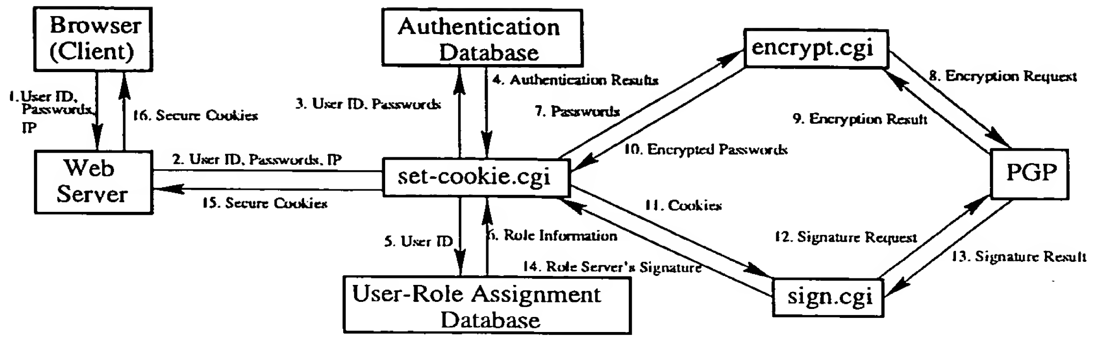


Figure 4: Creating Secure Cookies

Language [BJR98]) style notation for secure cookie creation. This diagram shows how we create a set of secure cookies for our implementation (refer to the left side of Figure 3).

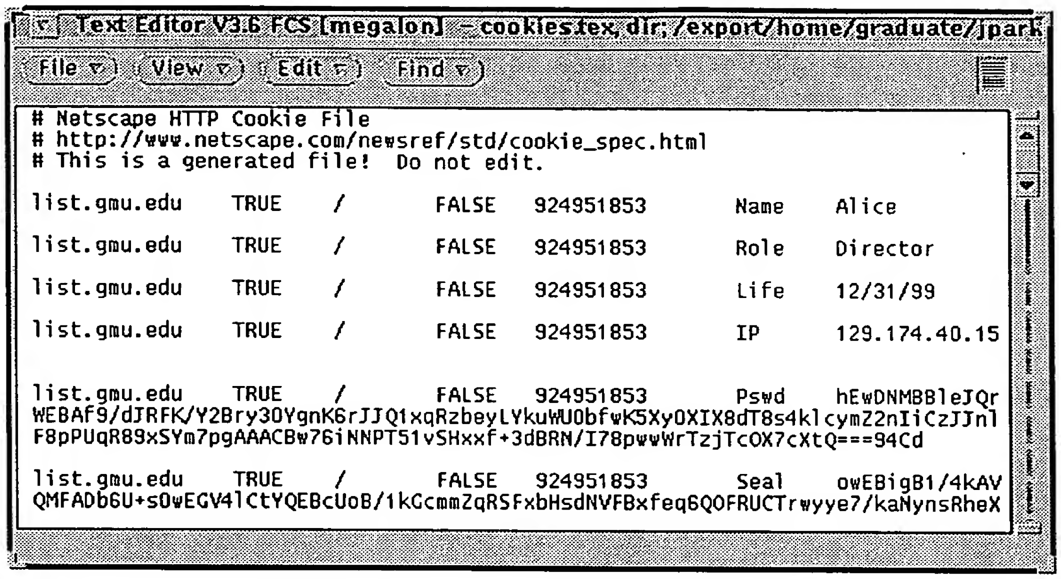
The Web server receives the request headers, which include the address to the "set-cookie.cgi" program on the server. The server translates the headers into environment variables and executes the program. The "set-cookie.cgi" program first retrieves the user ID and passwords, and the IP number of the client machine from the environment variable, REMOTE_ADDR. The program authenticates the user by comparing the user ID and passwords with the ones in the authentication database.[5] It then assigns the user to roles by matching the user ID and the corresponding roles from the URA (User-Role Assignment) database.

Subsequently, a subroutine for encryption is called to another CGI program (encrypt.cgi), which uses PGP to encrypt the passwords by the cookie-verifying Web server's public key. These encrypted passwords will be stored in the Pswd_Cookie by the "set-cookie.cgi" program. Then, the "set-cookie.cgi" program creates IP_Cookie, Pswd_Cookie, Name_Cookie, Life_Cookie, and Role_Cookie, giving each cookie the corresponding value: IP number of the client machine, encrypted passwords, user's name, lifetime of the cookie set, and assigned roles.

To support the integrity service of the cookies, the "set-cookie.cgi" program calls another CGI program (sign.cgi), which uses PGP to sign on the message digest with the role server's private key. The "set-cookie.cgi" then creates the Seal_Cookie, which includes the digital signature of the role server on the message digest of the cookies.

Finally, the Web server sends the HTTP response header, along with the cookies, back to the user's browser, and the cookies are stored in the browser until they expire. These secure cookies will be verified and used in the Web servers as described in the following subsections. Figure 5 is an actual snapshot of a set of secure cookies from our implementation that are stored in the user's machine after the cookies are generated by the cookie-issuing Web server. The contents of the cookies exactly reflect the ones presented in Figure 1. Each

[5]If the user already has an authentication cookie in a set of secure cookies, Web servers can use the authentication cookie for user authentication instead of authentication databases.


Text Editor V3.6 FCS [megalon] - cookies.tex, dir; /export/home/graduate/jpark
File
View
Edit
Find

```
# Netscape HTTP Cookie File
# http://www.netscape.com/newsref/std/cookie_spec.html
# This is a generated file!  Do not edit.

list.gmu.edu    TRUE    /       FALSE   924951853       Name    Alice

list.gmu.edu    TRUE    /       FALSE   924951853       Role    Director

list.gmu.edu    TRUE    /       FALSE   924951853       Life    12/31/99

list.gmu.edu    TRUE    /       FALSE   924951853       IP      129.174.40.15


list.gmu.edu    TRUE    /       FALSE   924951853       Pswd    hEwDNMBBleJQr
WEBAf9/dJRFK/Y2Bry30YgnK6rJJQ1xqRzbeyLYkuWU0bfwK5Xy0XIX8dT8s4klcymZ2nIiCzJJnl
F8pPUqR89xSYm7pgAAACBw76iNNPT51vSHxxf+3dBRN/I78pwwWrTzjTcOX7cXtQ===94Cd

list.gmu.edu    TRUE    /       FALSE   924951853       Seal    owEBigB1/4kAV
QMFADb6U+sOwEGV4lCtYQEBcUoB/1kGcmmZqRSFxbHsdNVFBxfeq6QOFRUCTrwyye7/kaNynsRheX
```

Figure 5: An Example of Secure Cookies Stored in a User's Machine

cookie has its corresponding domain, flag, path, security flag, expiration date, name, and value. The user's name, role, lifetime of the cookie set, IP number, encrypted passwords, and the digital signature of the cookie-issuing Web server on the cookies are stored in the corresponding cookies.

4.2 Secure Cookie Verification

Figure 6 is a collaborational diagram in UML style notation for secure cookie verification. This diagram shows how we verify (corresponding to the right side of Figure 3) the set of secure cookies that we generated in the previous subsection for our implementation. When Alice connects to a Web server (which accepts the secure cookies) in an RBAC-compliant domain, the connection is redirected to the "index.cgi" program. The related secure cookies are sent to the Web server and she is prompted by the HTML form to type in her user ID and passwords. The "index.cgi" program first uses the HTTP_COOKIE environment variable to retrieve the secure cookies (Name_Cookie, Life_Cookie, Role_Cookies, IP_Cookie, Pswd_Cookie, and Seal_Cookie) for the Web server. It then checks the validity of all the cookies. The two IP addresses, one from the IP cookie and the other from the environment variable, REMOTE_ADDR, are compared. If they are identical, then the host-based authentication is passed, and a hidden field "status" with the value of "IP-passed" is created to indicate that this stage was passed[6]. However, if the IP numbers are different, the user

[6]We used a hidden field to check the completion of the previous stage, which is passed on to the next program. This hidden field protects the pages from being accessed directly, skipping required verification steps, by a malicious user. For example, without this hidden field, a malicious user can access the pages

09451090 . 113099

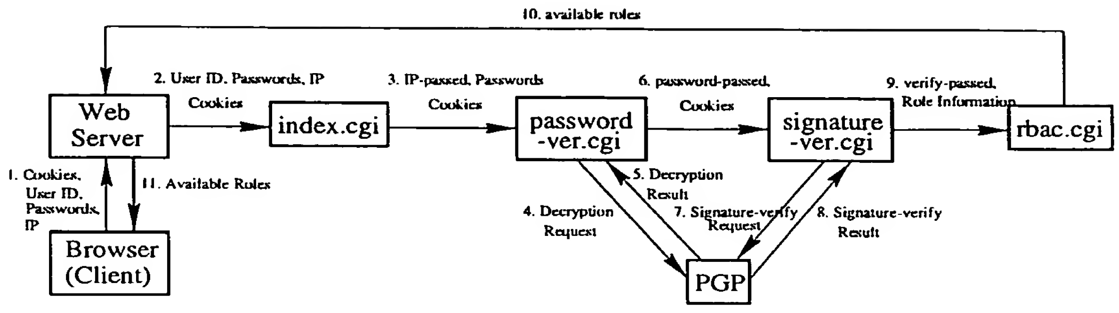

10. available roles
2. User ID, Passwords, IP Cookies
3. IP-passed, Passwords Cookies
6. password-passed, Cookies
9. verify-passed, Role Information
Web Server
index.cgi
password -ver.cgi
signature -ver.cgi
rbac.cgi
1. Cookies, User ID, Passwords, IP
11. Available Roles
Browser (Client)
4. Decryption Request
5. Decryption Result
7. Signature-verify Request
8. Signature-verify Result
PGP

Figure 6: Verifying Secure Cookies

is rejected by the server.

When the user submits her user ID and passwords to the server, the Web server translates the request headers into environment variables, and another CGI program, "password-ver.cgi," is executed. We used the POST method to send the information to the Web server and the ACTION field to specify the CGI program (password-ver.cgi) to which the form data are passed. The first thing the "password-ver.cgi" does is to check the hidden field "status" to see if the previous stage was successfully completed. If this is "IP-passed," the program decrypts the value of the Pswd_Cookie (encrypted user password) using the PGP with the Web server's private key, since it was encrypted with the Web server's public key by the role server. The program (password-ver.cgi) then compares the two passwords: one from the user and the other decrypted from the Pswd_Cookie. If they are identical, then the user-based authentication is passed, and a hidden field "status" with the value of "password-passed" is created to indicate that this stage was passed. However, if the two passwords are different, the user has to start again by either retyping the passwords or receiving new cookies from the role server.

After the password verification is completed, another CGI program, "signature-ver.cgi," is activated to check the integrity of the cookies. Like the other programs, it first checks the value of "status" passed on from the previous program, and it proceeds only if it shows the user has been through the password verification stage. If the value is "password-passed," then the program verifies the signature in the Seal_Cookie with the role server's public key using PGP. If the integrity is verified, it means that the cookies have not been altered, and a hidden field "status" with the value of "verify-passed" is created to indicate that this stage was passed and forwarded to the final program, "rbac.cgi." This program uses the role information in the Role_Cookie for role-based access control in the server as described in the following subsection. However, if the signature verification is failed, the user has to start again by receiving new cookies from the role server.

directly with forged cookies.

09451606.113099

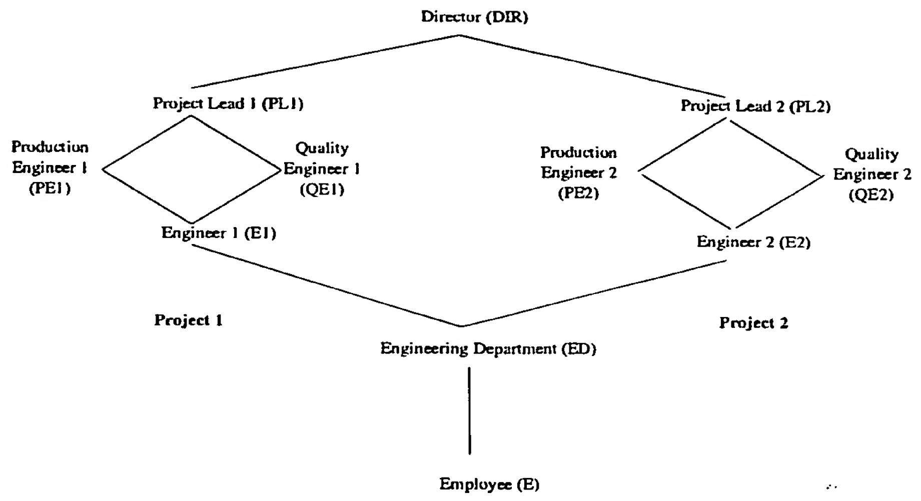


Figure 7: An Example Role Hierarchy

4.3 RBAC in the Web Server

After verifying all the secure cookies, the Web server allows the user, Alice, to execute transactions based on her roles, contained in the Role_Cookie, instead of her identity. In other words, the Web server does not care about the user's identity for authorization purposes. This resolves the scalability problem of the identity-based access control, which is being used mostly in existing Web servers. Furthermore, the Web server can also use a role hierarchy, which supports a natural means for structuring roles to reflect an organization's lines of authority and responsibility. Each Web server may have a role hierarchy defferent from that in other servers. In our implementation, we used a role hierarchy in the Web server, depicted in Figure 7. The location of RBAC-compliant Web servers is geographically free from that of the role server, since cookies can be issued by one Web server for use by others, regardless of their physical location.

If the "rbac.cgi" program in Figure 6 receives the value, "verify-passed," from the previous verification step, it means that the cookies have successfully passed all the verification stages, such as IP, passwords, and signature verification. Therefore, the Web server can trust the role information in the Role_Cookie, and uses it for role-based access control in the server.

Suppose Alice has the role DIR[7] in her Role_Cookie and she wants to access resources for PE1 - which require the PE1 role or roles senior to the PE1 role in the role hierarchy - in a Web server. First, she has to prove that the cookies she is presenting are genuine. To prove this, she has to go through all the verification steps: IP, passwords, and signature verification. She cannot jump ahead or skip any verification stage, as each program requires a hidden field, "status," from the previous stage. After the Web server has successfully

[7]Multiple roles can be stored in a Role_Cookie.

09451095.111303

completed all the verification steps, the "rbac.cgi" program retrieves the role information, DIR, from Alice's Role_Cookie, and shows all the available roles[8] based on the role hierarchy depicted in Figure 7. Since she wants access to the pages that require PE1's privilege, she chooses the PE1 role to activate it among her available roles. Then she has the permissions assigned to the PE1 role and the roles junior to PE1, such as E1, ED and E. Now, when Alice requests access to a particular page in the server, the page checks if her activated role, PE1, has permission to access the page.

The user can use any roles among her available roles by activating them. For instance, if Alice would activate PL1, then she would be allowed to access the pages, which require the PE1 role, since PL1 is senior to PE1 in the role hierarchy. However, if she were to activate E1, then she would not be allowed to access the pages, since E1 is junior to PE1. This supports least privileges, since only those permissions required for the tasks are turned on by activating the required role.

How then can the Web server protect the pages from being accessed by unauthorized users? Suppose a malicious user, Bob, has the role PE1 but wishes to access pages that require the PL1 role. He could change the value of his Role_Cookie so that it has PL1, or roles senior to PL1. He would go through the password verification stages, since he would be able to log in as Bob by using his own passwords. However, when his Seal_Cookie is being verified, there would be a problem, as the signature verification would fail. Therefore, he would not be allowed to move beyond this stage. On the other hand, he could try accessing the pages directly by typing the URLs. This would not be allowed, since each page checks to see if he has activated the required role, PL1, or roles senior to PL1. In other words, Bob is not allowed to access the pages, which require roles senior to his, because he cannot activate the senior roles, which are out of his available role range.

As a result, the Web server allows only users, who have gone through all the verification steps with the secure cookies (Name_Cookie, Life_Cookie, Role_Cookies, IP_Cookie, Pswd_Cookie, Seal_Cookie), to access the pages. This access also is possible only if the users have the required roles and activate them among their available roles based on the role hierarchy.

09451060.113099

5 Conclusions

In this paper, we have described how we implemented RBAC with role hierarchies on the Web using *secure cookies*. To protect the role information in the cookies, we provided security services, such as authentication, confidentiality, and integrity, to the cookies using PGP and CGI scripts in the Web servers. The cookie-issuing Web server creates a set of secure cookies including the user's role information, and other Web servers use the role information for RBAC with role hierarchies after cookie verification. This access control mechanism solves the scalability problem of existing Web servers. The use of secure cookies is a transparent process to users and applicable to existing Web servers and browsers.

[8] In this example, all the roles from E to DIR are available to Alice, since she has the senior most role in the role hierarchy.

6 Acknowledgements

This work is partially supported by the National Security Agency under its IDEA program.

References

[BJR98] Grady Booch, Ivar Jacobson, and James Rumbaugh. *The unified modeling language user guide*. Addison-Wesley, 1998.

[CDFT98] J. Callas, L. Donnerhacke, H. Finney, and R. Thayer. *OpenPGP massage Format*, November 1998. RFC 2440.

[FCK95] David Ferraiolo, Janet Cugini, and Richard Kuhn. Role-based access control (RBAC): Features and motivations. In *Proceedings of 11th Annual Computer Security Application Conference*, pages 241–48, New Orleans, LA, December 11-15 1995.

[Her96] Eric Herrmann. *CGI Programming with Perl 5*. Sams Net, 1996.

[KM97] David M. Kristol and Lou Montulli. *HTTP state management mechanism*, February 1997. RFC 2109.

[KM98] David M. Kristol and Lou Montulli. *HTTP state management mechanism*, July 1998. draft-ietf-http-state-man-mec-10.txt.

[Neu94] B. Clifford Neuman. Using Kerberos for authentication on computer networks. *IEEE Communications*, 32(9), 1994.

[RRSW97] C. Rigney, A. Rubens, W. A. Simpson, and S. Willens. *Remote Authentication Dial In User Service RADIUS*, April 1997. RFC 2138.

[RS98] E. Rescorla and A. Schiffman. *Security Extensions For HTML*, June 1998. draft-ietf-wts-shtml-05.txt.

[San98] Ravi Sandhu. Role-based access control. *Advances in Computers*, 46, 1998.

[SB97] Ravi Sandhu and Venkata Bhamidipati. The URA97 model for role-based administration of user-role assignment. In T. Y. Lin and Xiaolei Qian, editors, *Database Security XI: Status and Prospects*. North-Holland, 1997.

[SBC+97] Ravi Sandhu, Venkata Bhamidipati, Edward Coyne, Srinivas Ganta, and Charles Youman. The ARBAC97 model for role-based administration of roles: Preliminary description and outline. In *Proceedings of 2nd ACM Workshop on Role-Based Access Control*, pages 41–50. ACM, Fairfax, VA, November 6-7 1997.

[SCFY96] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, February 1996.

0945109 . 11309

[SNS88] J.F. Steiner, C. Neuman, and J.I. Schiller. Kerberos: An authentication service for open network systems. In *Proc. Winter USENIX Conference*, 1988.

[SP98] Ravi Sandhu and Joon S. Park. Decentralized user-role assignment for Web-based intranets. In *Proceedings of 3rd ACM Workshop on Role-Based Access Control*, pages 1–12. ACM, Fairfax, VA, October 22-23 1998.

[SR98] A. Schiffman and E. Rescorla. *The Secure HyperText Transfer Protocol*, June 1998. draft-ietf-wts-shttp-06.txt.

[WS96] D. Wagner and B. Schneier. Analysis of the SSL 3.0 protocol. In *Proceedings of the Second UNIX Workshop on Electronic Commerce*, November 1996.

[Zim95] Phillip R. Zimmermann. *The Official PGP User's Guide.* MIT Press, 1995.

09451090 . 113099